Eötvös Loránd Tudományegyetem

Természettudományi Kar

Horváth Ákos

Disztributív címkéző algoritmusok

BSc Szakdolgozat

Témavezető:

Kovács Erika Renáta

Operációkutatási Tanszék

Budapest, 2014

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani témavezetőmnek, Kovács Erikának, az általa nyújtott rengeteg segítségért, a rendszeres konzultációkért és végtelen türelméért. Túlzás nélkül állíthatom, hogy nélküle ez a dolgozat nem jöhetett volna létre.

Köszönettel tartozom még a családomnak, a páromnak, tanáraimnak, szaktársaimnak és barátaimnak – az ő olykor biztató, olykor kétkedő szavaik folyamatosan motiváltak, és így nagyban segítették ezen dolgozat létrejöttét.

Tartalomjegyzék

1. fejezet

Bevezető

Ebben a dolgozatban disztributív, gráfokkal kapcsolatos algoritmusokkal fogunk foglalkozni. Egy – gráfon értelmezett – algoritmust akkor nevezünk disztributívnak, ha az általa végzett számítások valamilyen értelemben lokálisak, vagyis a csúcsoknál helyileg történnek. Előfordulhat olyan információ, ami minden csúcsnál elérhető – például az algoritmus egy vagy több bemenete – , de a csúcsoknak a számítások elvégzése közben tipikusan kevés információjuk van a többi csúcsról.

Az algoritmusok számításainak elosztása, vagyis disztribúciója több szempontból is fontos. Egyrészt rengeteg alkalmazás létezik, ahol hasznos, ha egy nagy rendszer a hálózat egészére vonatkozó feladatokat felsőbb irányítás nélkül tud véghezvinni. Ilyen lehet például egy irodaépület belső, vezeték nélküli kommunikációs hálózata, de akár az internetes peer-to-peer adatcsere is. Másrészről a számítógépek fejlődésével – különösképp a többmagos processzorok megjelenésével – egy algoritmus részekre osztása, és a részek párhuzamos végrehajtása nagyban javíthat a teljesítményen, akkor is, ha azt egyetlen számítógép végzi, és fizikailag nincsenek szétszórva a részfeladatokat végző egységek. Mi ez utóbbi kérdéskörrel nem foglalkozunk.

Szeretnénk előrevetíteni, hogy két nagy részre bonthatóak a disztributív algoritmusok, az alapján, hogy milyen hálózaton futtatjuk őket. A hálózat lehet statikus, vagyis belátható időn belül nem változó (például egy úthálózat); vagy dinamikus, azaz időben változó – ennek számos típusa van, ezek közül néhányat később precízen definiálunk. Természetesen dinamikus hálózaton sem csak disztributív algoritmust lehet értelmezni, de mi ezekkel fogunk érintőlegesen foglalkozni.

1.1. Alapprobléma

Az alapvető feladat a legtöbb esetben az, hogy a bemenetből a csúcsok szomszédaikkal való kommunikáció segítségével kiszámoljanak maguk számára egy bizonyos információhalmazt. Ebben a dolgozatban leginkább – de nem kizárólag – az útkeresés témakörével szeretnénk foglalkozni. Ekkor célunk egy *routing table* konstruálása.

1.1.1. Definíció. *Egy v csúcsnál található, általában $(d, s, next_v(s))$ alakú információhármast* ***címkének*** *nevezünk, ahol s a célállomást, d az oda vezető út távolságát jelenti, $next_v(s)$ pedig megadja a következő lépést a cél felé vezető úton. Egy adott csúcsnál található címkék összességét* ***címkelistának*** *nevezzük. A gráf összes csúcsánál található címkelisták összességét* ***irányító táblázatnak*** *vagy* ***routing table-nek*** *nevezzük.*

Megjegyezzük, hogy egy címkének nem feltétlenül kell ilyen alakúnak lennie, még akkor sem, ha irányító táblázatot konstruálunk: a 4. fejezetben információnégyesek, illetve ötösök is elő fognak fordulni, ez általában függ az adott algoritmustól, illetve akár annak implementációjától.

1.1.2. Definíció. *Azt mondjuk, hogy egy v_0 csúcs* ***el tud irányítani*** *egy v_k csúcsba, ha a címkelistájában van olyan $(d, s, next_{v_0}(s))$ címke, ahol $s = v_k$, valamint minden $i \in \{1, \ldots, k-1\}$-re $v_i = next_{v_{i-1}}(v_k)$, és v_i címkelistájában is megtalálható v_k.*

Megjegyezzük, hogy az "el tud irányítani" kifejezés a gyakorlati alkalmazásokból ered, hiszen ezeknél tipikusan egy üzenetet – másnéven: csomagot – kell eljuttatni egy célállomásra. Az üzenet rendelkezik a célállomásra vonatkozó szükséges információkkal, és ez alapján az irányítótáblánk irányítja.

Most pedig lássuk az egyik legalapvetőbb disztributív címkéző eljárást, a Bellman-Ford algoritmust.

2. fejezet

Disztributív Bellman-Ford algoritmus

Adott egy $G(V, E)$ irányítatlan gráf $W : E \to \mathbb{R}^+$ súlyfüggvénnyel, valamint egy kitüntetett $s \in V$, a forráspont. Célunk megkonstruálni egy irányító táblázatot, ami alapján a gráf bármely csúcsából el tudunk irányítani a forráspontba legrövidebb úton.

2.0.3. Jelölés. Legyen **source** : $V \to V \cup \{\perp\}$ egy függvény. Egy $v \in V$ csúcsra source$(v) := v$, ha v a forráspont, source$(v) := \perp$ egyébként.

2.0.4. Jelölés. Jelöljük d_v-vel a v csúcs címkéjében található d távolságot.

2.0.5. Megjegyzés. Magát a gráfot nem soroljuk az inputok közé, mert a *gráfon* futtatjuk az algoritmust.

2.0.6. Megjegyzés. Ugyan az algoritmust egyetlen forrásponttal adjuk meg, vagyis mindenki egy kitüntetett csúcsba szeretne elirányítani legrövidebb úton, ugyanakkor az eljárás könnyen kiterjeszthető $1 < k \leq n$ forráspontra. Ekkor egy csúcs nem egyetlen címkét tárol a címkelistájában, hanem annyit, ahány forráspont van. Ilyenkor az algoritmus minden beérkező címkére megkeresi a címkében szereplő forráspontról lévő aktuális adatát, és azt frissíti.

Algorithm 1: Disztributív Bellman-Ford algoritmus egy forrásponttal

input : Minden $v \in V$ ismeri $|V| = n$-t, source(v)-t és a rá illeszkedő éleket, illetve a szomszédait.

output: Minden v csúcsnál egy $(d_v, s, \text{next}_v(s))$ alakú címke, ahol s a forráspont, d_v a forrásponttól való távolság, $\text{next}_v(s)$ pedig megadja az s-be vezető legrövidebb út következő állomását.

```
1  foreach v ∈ V do
2  |  next_v(s) := v ;
3  |  if source(v) = v then
4  |  |  d_v := 0
5  |  else
6  |  |  d_v := ∞
7  for i = 1, ..., n − 1 do
8  |  foreach v ∈ V do
9  |  |  if d_v < ∞ then
10 |  |  |  Elküldi az éppen aktuális (d_v, s, next_v(s)) címkéjét a szomszédainak
11 |  |  foreach Minden w ∈ V csúcstól kapott (d_w, s, next_w(s)) címkére do
12 |  |  |  if d_w + W({v, w}) < d_v then
13 |  |  |  |  (d_v, s, next_v(s)) := (d_w + W({v, w}), s, w)
14 foreach v ∈ V do
15 |  return (d_v, s, next_v(s))
```

2.0.7. Megjegyzés. Az algoritmus néhány egyszerű módosítással irányított gráfon is működik, ekkor a nemnegatív élfüggvény helyett a gráf negatív köröktől való mentességét kell feltenni. Ugyanakkor ez a feltétel elhagyható, abban az értelemben, hogy az algoritmus outputjából könnyedén ellenőrizhető, hogy található-e a gráfban negatív kör. Minden $\{u, v\}$ élre ellenőriznünk kell, hogy $d_u + W(\{u, v\}) < d_v$ teljesül-e. A gráf akkor és csak akkor tartalmaz negatív kört, ha ez a feltétel valamelyik élre teljesül. Ekkor természetesen az output irányító táblái **nem** lesznek alkalmasak a legrövidebb úton való irányításra.

2.1. Bellman-Ford változó hálózatban

Ebben a részben azt fogjuk röviden megvizsgálni két példán keresztül, hogy hogyan reagál a Bellman-Ford algoritmus a gráf megváltozására. Ez a kérdés alapvető fontosságú, hiszen ezen múlik, hogy az algoritmusunk – jelenlegi állapotában – alkalmas-e például egy számítógéphálózat

kommunikációját irányítani. Az ebben a részben szereplő példákat [1] motiválta.

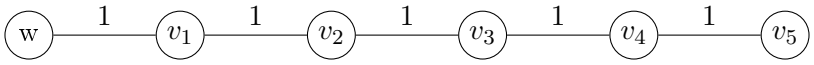
Először is tekintsük az algoritmus azon módosított verzióját, ahol minden csúcs egy különálló forráspont, vagyis minden csúcs n darab címkét tárol a címkelistájában – ekkor az algoritmus lefuttatása után bárhonnan bárhová el tudunk irányítani legrövidebb úton. Továbbra is tegyük fel, hogy az élfüggvény pozitív.

Tegyük fel továbbá, hogy nem vetünk véget az algoritmusnak, vagyis a ciklusunk a végtelenségig fut és a csúcsok egymással előre megadott időközönként címkelistát cserélnek. Ekkor, ha a gráf nem változik, akkor egy idő után már nem fog változni a címkelista egyik csúcsnál sem, ilyenkor azt mondjuk, hogy az algoritmus **stabilizálódott**.

Módosítsuk továbbá az algoritmust annyiban, hogy egy csúcs ne csak akkor változtasson a már meglévő címkéjén, ha annál jobbat kap, hanem mindig vesse össze az egy adott forráspontra vonatkozó beérkező adatokat, és azok közül válassza ki a legjobbat. Ez ugyan több lokális erőforrást igényel, de szükséges, amennyiben a gráf megváltozik.

2.1.1. Egy csúcs hozzáadása

Miután beállt a fent leírt állapot, adjunk hozzá a hálózathoz egy új w csúcsot, mely néhány éllel csatlakozik a már meglévő csúcsokhoz. Minden csúcs, ami w-vel szomszédos, a következő körben felveszi w-t a címkelistájába a becsült távolságokkal. A w csatlakozását követő második körben w szomszédainak szomszédai is mind tudomást szereznek w-ről, és így tovább. A gráf minden csúcsa legrövidebb úton fog tudni irányítani w-be, és az összes többi csúcsba legfeljebb n kör múlva (ahol n az eredeti gráf csúcsainak száma). Ez biztosan igaz, hiszen az eredeti algoritmus kezdeti bemenetéből indulva (amikor minden csúcsnál minden távolság ∞) véget érne n kör alatt. Nézzünk egy példát arra, amikor valóban n körig tart a stabilizáció.



A fenti példán jól látható, hogy v_5 valóban az 5. körben szerez tudomást w-ről, és az 5. kör után tud minden csúcsba legrövidebb úton irányítani. Tetszőlegesen nagy n-re adható hasonló példa, ahol a gráf egyetlen út.

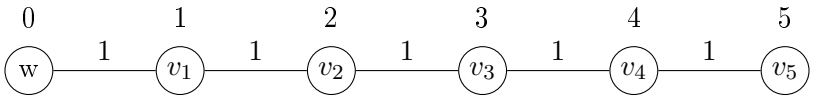
Egy él hozzávétele rendkívül hasonló, ezért arra nem térnénk ki. Ebben az esetben szintén legfeljebb n kör alatt stabilizálódik az algoritmus.

Levonhatjuk tehát a következtetést, hogy a Bellman-Ford algoritmus jól reagál egy csúcs vagy egy él megjelenésére, amennyiben az új élek költségei pozitívak. Ez irányított esetben is hasonlóképpen működik, akkor azt kell feltennünk, hogy nem keletkezik negatív kör a gráfban.

Az algoritmus stabilizálódása viszonylag rövid idő alatt megtörténik, és a stabilizáció előtt sem történhet félreirányítás, legfeljebb nem a legrövidebb úton halad majd a csomagunk.

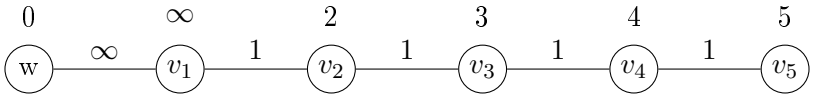
2.1.2. Egy csúcs eltávolítása

Egy csúcs (vagy egy él) eltávolítása a gráfból sokkal súlyosabb következményekkel jár. Egy példán keresztül mutatjuk be, hogy mennyire elromolhat az algoritmus egy csúcs elvétele esetén. A kezdeti állapotban a még változatlan gráfot mutatjuk az algoritmus stabil állapotában. A csúcsok fölé feljegyeztük, hogy a címkéjük szerint w-be vezető legrövidebb út milyen hosszú.

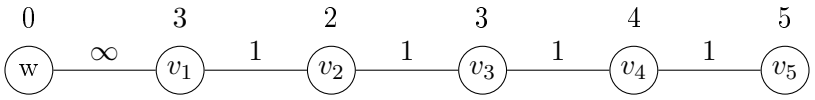


A következő pillanatban vegyük el w-t és a hozzá tartozó élet. Megjegyezzük, hogy ez ugyanolyan eredményt szül, mintha csak az élet vennénk el, így a két esetet egy példán keresztül mutatjuk be.

Megjegyezzük továbbá, hogy feltételezzük, hogy ilyen helyzetben az él megszűnését v_1 oly módon érzékeli, mintha abba az irányba ∞ hosszú út vezetne w-be, és a címkéjét is ennek megfelelően módosítja. A megszűnés utáni pillanat, ahol a csúcsok még nem kommunikáltak egymással, tehát így írható le:

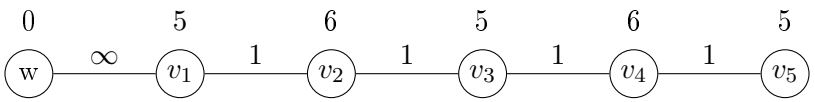


A következő kommunikáció során v_1 kétféle információt tud w-ről: egyrészről érzékeli, hogy egy ∞ hosszú útja van oda (másszóval nem tud oda irányítani), másrészről v_2-től olyan információt kap, hogy arra tud 2 hosszú úton menni w-be. Így tehát v_1 frissíti a w-re vonatkozó címkéjét $(w, 3, v_2)$-re.



Ezután v_2 mindkét szomszédjától azt az információt kapja, hogy belőlük 3 hosszú út vezet w-be. Mivel v_2 ezeknél rövidebb útról nem kap információt, ezért frissíti a címkéjében a w-be vezető út hosszát 4-re.

Az ez utáni körben hasonló meggondolásból v_1 és v_3 frissítik a w-be vezető út hosszát 5-re. Az ezt követő körben v_2 6-ra módosít, ekkor már v_4 is kénytelen 6-ra módosítani.



Innen már bizonyára látszik, hogy mi fog történni: a csúcsok szép lassan egyesével növelik a w-ra vonatkozó távolságukat, ez a növelés pedig sosem áll meg (amennyiben a gráf nem változik). Ezért is hívják ezt **a végtelenig számolás problémájának**.

A végtelenig számolás rengeteg gondot vet fel. A legszembeötlőbb, hogy útjára tudunk bocsájtani egy w-be címzett csomagot, és az algoritmus ezt a csomagot a végtelenségig fogja irányítani anélkül, hogy az valaha is célbaérhetne. És ez nem feltétlenül csak a w-be címzett csomagokkal lehet így: tegyük fel, hogy nem a fenti példagráfon vagyunk, és egy csúcs eltávolításával több komponensre szakad a gráfunk. Ekkor semelyik komponens nem fogja megtudni egyik másikról sem, hogy oda már nem tud irányítani.

Gyakorlati probléma továbbá a folyamatos számolás. A disztributív hálózatoknál általában valamilyen korlátot szoktunk szabni a címke illetve az időegység alatt elcserélhető információ méretének, de a lokális számítás erőforrásigényeit sem érdemes figyelmen kívül hagyni, hiszen a csúcsaink lehetnek olyan független eszközök, amik csak korlátozott, belső áramforrással rendelkeznek – például mobiltelefonok vagy rádiókészülékek. Az ilyen eszközök számára rendkívül megterhelő is lehet a folyamatos, véget nem érő számolás. Gondoljunk csak arra az imént említett példára, ahol w elvételével a gráf komponensekre szakad: ekkor egy csúcs az elérhetetlen komponens méretével megegyező számú címkét fog minden körben feleslegesen frissíteni.

A végtelenig számolás problémája abból adódik, hogy egy adott csúcs nem tudja megmondani, hogy ő rajta van-e azon az úton, amit egy másik csúcs javasol neki. A fenti példagráfban javíthattunk volna a helyzeten, ha a szomszédoktól kapott *next* pointereket is figyelembe vesszük, de ez komplikáltabb gráfokban nem feltétlenül jelent megoldást.

2.1.3. A végtelenig számolás kezelése a gyakorlatban

Számtalan olyan apró módosítása létezik a Bellman-Ford algoritmusnak, amik így vagy úgy valamennyire kezelik ezt a problémát, de a megoldások hatékonysága általában nagyban függ a konkrét gyakorlati alkalmazástól. Mi most a részletekben való elmélyedés nélkül ismertetünk néhány lehetséges megoldást, részletes leírás [2]-ben található.

A végtelen megválasztása. A legalapvetőbb megoldás a végtelen gyakorlati megválasztásában rejlik: a valóságban mindig létezik egy maximális távolság, ami nem létezhet a gráfunkban. Élsúlyozatlan esetben $|E| + 1$ például ilyen. Kiegészíthetjük az algoritmusunkat úgy, hogy ezt a számot választjuk "végtelennek", és amennyiben egy v csúcs ezt a számot írná valamelyik címkéjébe, úgy az abban a címkében szereplő csúcsot tekintsük elérhetetlennek. Könnyen látható, hogy ezen módszer alkalmazásával nem fogunk ténylegesen végtelenig számolni, és az algoritmusunk idővel stabilizálódni fog, azonban nagyon hosszú számolások még így is előfordulhatnak.

Split horizon with poisoned reverse. A fent említett megoldás felgyorsítására alkalmazható eljárás lényege a következő: amikor egy v csúcsúnk egy $(d, s, next_v(s))$ címkéjével kapcsolatos információt továbbít a szomszédainak, akkor $next_v(s)$ felé azt közvetíti, hogy s számára elérhetetlen. Ezzel a gyakorlatban számos végtelenig számolás megelőzhető, például a fent bemutatott példán nem következne be, ugyanakkor nem szolgál általános megoldásul.

Kiváltott frissítések. Ez a módszer a fent ismertetett két eljárás kiegészítéseként célravezető. A gyakorlati alkalmazásokban a csúcsok közti kommunikáció történhet viszonylag ritkán (például harminc másodpercenként), ezt most szokásos frissítésnek fogjuk nevezni. Ezen módszer lényege, hogy bármikor, amikor egy csúcs változtatást hajt végre a címkelistájában, erről azonnal egy különleges üzenetet (egy kiváltott frissítést) küld a szomszédainak, nem megvárva a következő szokásos frissítést. Amennyiben ez a szomszédok címkelistájában frissítést eredményez, a szomszédok is kiváltott frissítést küldenek a szomszédaiknak, és így tovább. Ez a módszer ritkán frissülő hálózatokban alkalmazható, ugyan a problémát nem szünteti meg teljesen, a stabilizációt felgyorsítja.

3. fejezet

Könnyű problémák dinamikus hálózatokban

Ahogy azt az előző fejezetben láthattuk, az útkeresés egy elsőre nehéz probléma egy változó hálózatban, ezért vizsgáljunk meg néhány egyszerűbb kérdést. Mielőtt azonban ezt megtehetnénk, be kell vezetnünk néhány alapfogalmat.
Elég csak arra gondolnunk, hogy már a *dinamikus hálózat* sem egy egyértelműen definiált dolog. Valamiféle változó gráfról beszélünk, de rengeteg kérdés nyitva áll előttünk: csak az élek változnak, vagy a csúcsok is? Véletlenszerűen változnak, vagy valamilyen rendszert követnek? Milyen gyakran változnak?

3.1. Alapfogalmak

Az egyszerűség kedvéért egyelőre tegyük fel, hogy a csúcshalmazunk rögzített, és az élhalmazunk a változó. Egy ilyen dinamikus gráfot a következőképpen lehet leírni.

3.1.1. Definíció. *Legyen $G(V, E)$ egy dinamikus gráf, ahol V a rögzített csúcshalmaz, $E : \mathbb{N}^+ \to 2^{V \times V}$ pedig a dinamikus élfüggvény, mely minden $r \in \mathbb{N}^+$ körben megadja a pillanatnyi élhalmazt, $E(r)$-t.*
Az r. kör alatt azt az időintervallumot értjük, ami a kitüntetett időpillanatok – vagyis a gráf megváltozásának időpontjai – közül az $r-1$.-től az r.-ig tart.
Jelöljük $G(r)$-rel $G(V, E(r))$-t. Az ilyen dinamikus gráfokat szokás ***fejlődő gráfoknak*** *is nevezni (evolving graph).*

3.1.1. Ellenfél modellek

Ellenfél (*adversary*) modellnek nevezzük a módszert, ami alapján meghatározzuk a dinamikus élfüggvényt. **Adaptív legrosszabb-eset ellenfélnek** (*adaptive worst-case adversary*), röviden adaptív ellenfélnek nevezzük azt a modellt, ami "menet közben" generálja a gráfot. $E(r)$ kiválasztása függhet a csúcsok $r-1$. pillanatbeli állapotától, vagy akár bármelyik azt megelőző állapotától is. **Feledékeny legrosszabb-eset ellenfélnek** (*oblivious worst-case adversary*), röviden feledékeny ellenfélnek nevezzük azt a módszert, ami előre meghatározza az élfüggvényt, így a dinamikus hálózat alakulása nem függ az algoritmus végrehajtásától vagy a csúcsok pillanatnyi állapotaitól.

Végül pedig **véletlengráf ellenfélnek** (*random-graph adversary*) nevezünk egy olyan modellt, ami valamilyen valószínűségi eloszlás alapján választja ki $E(r)$-eket. Ez valójában nem egy ellenfél. Speciális esete, amikor minden $r+1$. körben ($r \geq 1$) minden $e \notin E(r)$ p valószínűséggel bekerül $E(r+1)$-be, minden $e \in E(r)$ pedig q valószínűséggel nem kerül bele $E(r+1)$-be. Amennyiben $q = 1-p$, úgy egy Erdős-Rényi véletlengráfot kapunk.

3.1.2. Dinamikus átmérő

Egy dinamikus gráfon futtatott algoritmus számításainak hatékonyságát befolyásoló tényezők közül alapvető fontosságú a dinamikus átmérő. Ennek definiálása előtt lássunk először egy másik fogalmat.

Tegyük fel, hogy egy adott gráfban minden csúcs rendelkezik egy egyedi információval. Szeretnénk minden csúcstól minden csúcshoz eljuttatni ezeket az információkat. Ehhez a következő egyszerű eljárást hajtjuk végre: minden csúcs minden élén elküldi az egyedi információt. Minden csúcs feljegyez és a későbbi körökben az összes élén továbbküld minden információt, amit kap. Az információk küldését addig folytatjuk, amíg minden csúcs meg nem kap minden információt.

3.1.2. Elnevezés. A fent leírt eljárást **elárasztásnak** (*flooding*) nevezzük.

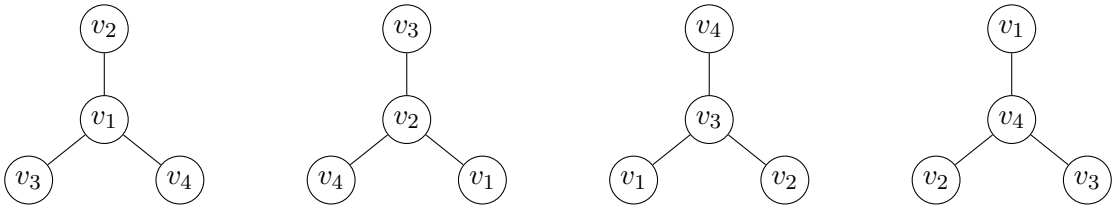
3.1.3. Definíció. *Egy dinamikus gráfban egy tetszőleges $t \in \mathbb{N}$ körben indítsunk el egy elárasztást. Legyen $D \in \mathbb{N}$ a legkisebb olyan, hogy a $t+D$. körben az elárasztás már befejeződött. Ekkor D a gráf t-hez tartozó* ***dinamikus átmérője****.*

A dinamikus átmérő egy természetes alsó határa annak, hogy mennyi időre van szükségünk ahhoz, hogy egy információt eljuttassunk a hálózat minden csúcsához. A következő példán keresztül megmutatjuk, hogy a pillanatnyi átmérő – vagyis a pillanatnyi $G(r)$ gráfok átmérője – miért nem szerencsés mérőszáma az információ terjedésének dinamikus hálózatban.

Vegyünk egy csillag gráfot, ahol egy csúcs a gráf középpontja, aki minden más csúccsal össze van kötve, az összes többi csúcs pedig csak a középponttal van összekötve, a költségfüggvény

legyen azonosan 1. A csúcsokat jelöljük v_i-vel, $i \in \{1, \ldots, n\}$. Kezdetben v_1 legyen középen, az éleket pedig úgy változtassuk, hogy minden $1 \leq t \leq n-1$ időben v_i kerüljön középre. Ha v_1 ismét középre kerül, akkor a folyamatot újrakezdjük.

3.1.4. Példa. A fent leírt példa $n = 4$ esetében az alábbi pillanatnyi gráfok ismétlődnek egymás után.



Akár a v_n-től, akár a v_{n-1}-től indított információt tekintjük, $n-1$ körre szükségünk lesz, mire az minden csúcsot elér, így a gráf dinamikus átmérője $n-1$, pedig a pillanatnyi gráfok átmérője mindig 2.

3.1.3. T-intervallum összefüggőség

A statikus gráfokhoz hasonlóan a dinamikus esetben is fontos feltétel az összefüggőség. A kérdést sok szempontból meg lehet közelíteni, mi ezek közül a [4]-ben definiált T-intervallum összefüggőséget választottuk. A dinamikus gráfot a fentebb ismertetett módon értelmezzük.

3.1.5. Definíció. *Azt mondjuk, hogy egy $G(V, E)$ dinamikus gráf **T-intervallum összefüggő** egy $T \geq 1$-re, ha minden $r \in \mathbb{N}$-re a statikus $G_{r,T} := (V, \bigcap_{i=r}^{r+T-1} E(r))$ összefüggő.*

A gráfot ∞-intervallum összefüggőnek nevezzük, ha létezik egy statikus, összefüggő $G' = (V, E')$ gráf, melyre igaz, hogy minden $r \in \mathbb{N}$-re $E' \subseteq E(r)$.

A ∞-intervallum összefüggőség tehát azt jelenti, hogy létezik egy statikus, összefüggő részgráfja a dinamikus gráfnak, amely végig változatlan marad. Ilyen lehet például egy feszítőfa.

A másik speciális eset az 1-intervallum összefüggőség, ami egybevág a pillanatnyi összefüggőség fogalmával. A fejezet további részében legfőként ezzel fogunk foglalkozni, de megemlítünk majd $T \geq 1$ eseteket is.

3.2. Problémák és feladatok dinamikus hálózatokban

Ebben a részben dinamikus hálózatokban felmerülő alapproblémákkal és azok lehetséges megoldásaival foglalkozunk. A dinamikus hálózatot az előző részben definiált módon értjük (vagyis statikus ponthalmazzal), egy adaptív ellenféllel. Feltesszük továbbá az 1-intervallum összefüggőséget. Ezen kívül más feltételt nem szabunk. **Tokennek** valamilyen egyedi információcsomagot nevezünk. Ebben a részben főleg a következő problémákkal foglalkozunk:

Számlálás. Azt mondjuk, hogy egy disztributív algoritmus megoldja a számlálás (*counting*) problémáját, ha egy n csúcsú dinamikus gráfon futtatva minden csúcsnál véges időn belül megáll a számítás, és minden csúcs outputja n.

k-igazolás. Közeli kapcsolatban áll a számlálással. A k-igazolásnál (*k-verification*) minden csúcs megkapja k-t, mint inputot, célunk pedig, hogy minden csúcs eldöntse $k \geq n$ igazságtartalmát, természetesen itt is véges időn belül leálló számítással.

k-token terjesztés. Ebben a problémában van k darab különböző tokenünk, és a kezdeti állapotban minden token pontosan egy csúcsnál található meg. Egy csúcsnál lehet több token, és nem minden csúcsnál kell tokennek lennie. Azt mondjuk, hogy egy disztributív algoritmus megoldja ezt a problémát, ha minden csúcsnál megáll, az output pedig mindenhol a k token lesz (vagyis a tokenek által tartalmazott információk összessége). Az, hogy a csúcsok ismerik-e k-t, a feladattól függ.

Mindenhonnan-mindenhova token terjesztés. A k-token terjesztés egy speciális esete, itt minden csúcs pontosan egy tokennel indul. A csúcsok nem ismerik $k = n$-t, de azt tudják, hogy minden csúcsnak van egy egyedi tokenje.

k-bizottság választás. Ebben a feladatban a csúcsok csoportokba, ún. bizottságokba osztják magukat. Minden bizottságnak van egy *bizottságazonosítója*, és a célunk az, hogy egy k inputból kiindulva minden csúcs visszaadja a saját bizottságazonosítóját oly módon, hogy a következő két feltétel teljesüljön: 1) minden bizottság mérete legfeljebb k és 2) amennyiben $k \geq n$, akkor csak egy bizottság van, amely tartalmaz minden csúcsot.

Token továbbítás. A **továbbítás** (*forwarding*) a feladat nevében arra utal, hogy a tokent nem lehet lemásolni. Az egész gráfban egyetlen token van, és minden csúcs tudja magáról, hogy nála van-e a token. Ha nála van, akkor eldöntheti, hogy magánál tartja még egy körig, vagy továbbítja valamelyik szomszédjának. A célunk lehet az, hogy a token bejárja az összes csúcsot, de akár az is, hogy egy kiválasztott csúcstól eljusson egy kiválasztott csúcsig.

Az ellenfelünk erejét demonstrálandó belátjuk a következő állítást.

3.2.1. Állítás. *Adaptív ellenféllel szemben a fent definiált dinamikus gráfban az említett feltételekkel a token továbbítás feladata nem megoldható, ha $n \geq 3$.*

Bizonyítás. Jelöljük v_0-lal azt a csúcsot, ahonnan a token indul, v_1-gyel pedig egy tetszőleges olyan csúcsot, ami nem a célállomás ($n = 3$ esetén ilyen mindig van). Az adaptív ellenfelünk v_0-t összeköti v_1-gyel, v_1-et pedig a gráf összes csúcsával. Így az 1-intervallum összefüggőség teljesül,

v_0-nak pedig két lehetősége van: vagy megtartja a tokent, vagy továbbítja v_1-nek. Ha megtartja, akkor az ellenfelünk nem változtat a gráfon, ha továbbíttja, akkor az ellenfelünk felcseréli v_0-t és v_1-et.

Könnyen látható, hogy a fenti példában a token csak v_0-t és v_1-et járja be, a gráf további részeibe sohasem jut el. □

Egy adaptív ellenféllel szemben tehát a csomagtovábbító algoritmusok esélytelenek, legalábbis ha nem teszünk fel erősebb összefüggőségi feltételeket. Viszont az összes többi említett probléma, ahol a tokenek másolhatók, megoldható. Amennyiben megengedünk "nagy" üzenetméreteket, úgy némelyik feladat egészen rövid idő alatt megoldható.

3.2.2. Állítás. ([4]) *A számlálás és a mindenhonnan-mindenhova token terjesztés feladatok megoldhatóak $\mathcal{O}(n)$ körben, ha a gráf 1-intervallum összefüggő és a csúcsok közti üzenetek $\mathcal{O}(n \cdot \log n)$ méretűek.*

Bizonyítás. □

A bizonyítás lényege, hogy minden csúcs minden körben minden általa ismert információt továbbít minden szomszédjának.

A dolgozat következő részében olyan megoldásokat szeretnénk találni, melyek $\mathcal{O}(\log n)$ méretű üzeneteket használnak.

3.2.1. A k-bizottság eljárás

Ahhoz, hogy 1-intervallum összefüggő dinamikus gráfokon végrehajtsuk a k-bizottság eljárást, feltesszük, hogy van egy kitüntetett vezércsúcs, aki a többi csúcsok közül néhányat meghív a saját bizottságába. Természetesen valójában nem választunk ki előre vezércsúcsot, erre visszatérünk később.

Az eljárás k ciklusból áll, mindegyik ciklus két fázisból:

Felmérő fázis. Ez a fázis $k-1$ körig tart. Minden körben minden csúcs továbbítja minden szomszédjának az általa már ismert csúcsok közül annak az azonosítóját, ami a legkisebb azok közül, akik még nem tagjai egyik bizottságnak sem. Kezdetben minden csúcs a saját azonosítóját küldi tovább, ha nem tagja bizottságnak, illetve a speciális $\perp$ jelet, ha igen. Megjegyzik a legkisebb értéket, amit kapnak a fázis során, és onnantól azt küldik tovább.

Meghívó fázis. A vezető kiválasztja a legkisebb azonosítót, amit eddig megkapott, és elküld egy üzenetet, amiben az adott azonosítójú csúcsot meghívja a bizottságába. Az üzenet tartalmazza a meghívó és a meghívott azonosítóit, és minden csúcs ezt az üzenetet továbbítja $k-1$ körön át. A fázis végén a meghívott csúcs csatlakozik a vezető bizottságához.

A k. ciklus végén minden csúcs, aki csatlakozott bizottsághoz, a vezetőjének az azonosítóját adja vissza bizottságazonosítóként. Aki nem csatlakozott egyetlen bizottsághoz sem, az a saját azonosítóját adja vissza bizottságazonosítóként.

Mivel valójában nem választunk ki előre egy vezetőt, kezdetben *minden* csúcs vezetőnek gondolja magát, egészen addig, amíg nem kapnak egy náluk kisebb azonosítót. Ekkor átváltanak a nem-vezető szerepre, és ezután csatlakozhatnak más vezetők bizottságaihoz. Ha egy csúcs egyszer csatlakozott egy bizottsághoz, akkor ezen már nem fog változtatni.

3.2.3. Tétel. ([4]) *A fent leírt eljárás megoldja a k-bizottság választásának problémáját $\mathcal{O}(k^2)$ kör alatt.*

Bizonyítás. □

Megjegyezzük, hogy a számlálás probléma későbbi megoldásához annyiban kell módosítani az eljárást, hogy minden csúcs jegyezzen fel minden azonosítót, amiről tudomást szerez. Ez az eljárás futásidejét nem változtatja nagyban, viszont $k \geq n$ esetben a k-bizottság feladat megoldása után minden csúcs ismerni fogja az összes csúcs azonosítóját, így a következő részben leírt módon megoldható a számlálás problémája.

3.2.2. Számlálás a k-bizottság alapján

Ebben a részben feltesszük, hogy meg tudjuk oldani a k-bizottság problémát, és megmutatjuk, hogy ebből hogyan hajthatjuk végre a számlálást és a token terjesztést.

Tegyük fel tehát, hogy a kezdeti állapotunk a k-bizottság feladat egy megoldása: minden csúcsnak van bizottságazonosítója, legfeljebb k csúcsnak ugyanaz a bizottságazonosítója, és ha $k \geq n$, akkor minden csúcsnak ugyanaz a bizottságazonosítója.

Ekkor $k \geq n$ igazságtartalmának eldöntése ekvivalens annak az eldöntésével, hogy egynél több bizottság van-e. Hiszen ha $k \geq n$, akkor biztosan egyetlen bizottság van, ha pedig $k < n$, akkor egynél többnek kell lennie. Ezáltal a csúcsok el tudják dönteni $k \geq n$-et úgy, hogy végrehajtanak egy eljárást, ami ellenőrzi a bizottságok számát.

A k-igazoló eljárás. Minden $v \in V$ csúcshoz tartozik egy x_v változó, aminek a kezdeti értéke 1. Amíg $x_v = 1$, v minden körben elküldi a bizottságazonosítóját minden szomszédjának. Ha v egy, a sajátjától különböző bizottságazonosítót kap, vagy a speciális $\perp$ jelet, akkor $x_v := 0$, és innentől v minden körben a $\perp$ jelet küldi el a szomszédainak. Az eljárás k kör után leáll, és minden v csúcs visszaadja az x_v változójának az értékét.

3.2.4. Lemma. *Tegyük fel, hogy a csúcsok eredeti állapotai a k-bizottság választás egy kimenetének feleltek meg. Ekkor a k-igazoló eljárás végén akkor és csak akkor ad vissza minden csúcs 1-et, ha $k \geq n$.*

Bizonyítás. Először tegyük fel, hogy $k \geq n$. Ekkor minden csúcs ugyanahhoz a bizottsághoz tartozik, így soha senki nem kap a saját bizottságazonosítójától különböző bejövő adatot, így a kezdeti $x_v = 1$ végig változatlan marad minden $v \in V$-re.

A másik irány bizonyításához a lemma állításánál kicsit többet látunk be. Megmutatjuk, hogy ha $k < n$, akkor minden csúcs 0-t ad vissza az eljárás végén.

Tegyük fel tehát, hogy $k < n$. Megmutatjuk, hogy az i. körben legalább i darab olyan v csúcs van, melyre $x_v = 0$. Vegyünk egy vágást a gráfban, az egyik oldalon legyen egy kitüntetett bizottság minden olyan v csúcsa, akire $x_v = 1$, a másik oldalon pedig minden másik csúcs. Az 1-intervallum összefüggőségből következően a vágás tartalmaz legalább egy élt, aminek az egyik végén egy u csúcs van a kitüntetett bizottságból, amire $x_u = 1$. Ez az u csúcs azonban olyan információt is fog kapni, ami nem egyezik meg a saját bizottságazonosítójával, így x_u-t 0-ra állítja, és a így az olyan csúcsok száma, melyekre $x_v = 0$, eggyel nőtt.

Ez minden bizottsággal eljátszható, mint kitüntetett bizottság. Mivel eredetileg minden bizottságban legfeljebb k csúcs van, ezért k kör után minden bizottság minden tagja 0-t fog adni visszatérési értéknek. □

A számlálást a következő módon oldjuk meg. Először végrehajtjuk a k-bizottság eljárást $k = 1$-re, utána az így kapott kimenettel a k-igazoló eljárást. Ha $k < n$, akkor k-t megduplázzuk és előről kezdjük. Ha $k \geq n$, akkor visszaadjuk n-et. Ezt meg tudjuk tenni, hiszen ebben az esetben minden csúcs ismeri minden csúcs azonosítóját, így csak meg kell számolniuk, hogy hány egyedi azonosítót jegyeztek fel a címkelistájukba.

Megjegyezzük, hogy ugyanezzel az eljárással megoldható a mindenhonnan-mindenhova token terjesztés: ehhez egyszerűen csak csatolnunk kell a tokeneket minden egyedi azonosítóhoz. Mivel minden csúcs megkap minden egyedi azonosítót, így minden csúcs megkap minden tokent is.

Az eddig leírtakat a következő tételben foglaljuk össze.

3.2.5. Tétel. (Kuhn, Lynch, Oshman, [4]) *Az előző részben ismertetett k-bizottság eljárás az ebben a részben leírt k-igazoló eljárással együtt $\mathcal{O}(n^2)$ körben megoldja a számlálás és a mindenhonnan-mindenhova token terjesztés feladatát.*

További eredmények. A [4]-es forrásban a szerzők kiterjesztik a fent ismertetett eljárást $T > 1$ esetre. Belátják, hogy a számlálás és a mindenhonnan-mindenhova token terjesztés megoldható $\mathcal{O}(n + \frac{n^2}{T})$ körben. Foglalkoznak továbbá azzal is, hogy milyen eredmény érhető el, ha T nem ismert a csúcsok számára. Ekkor mindkét probléma megoldható $\mathcal{O}(\min\{n^2, n+n^2 \cdot \frac{\log n}{T}\})$ körben.

4. fejezet

Lenzen és Patt-Shamir címkéző algoritmusa

A dinamikus hálózatokra tett kitekintés után térjünk vissza a hagyományos, statikus gráfokon értelmezett disztirbutív algoritmusokhoz. Ebben a fejezetben bemutatom Christoph Lenzen és Boaz Patt-Shamir randomizált címkéző algoritmusát.

Alapötlet. Az algoritmus intuíciója azon a megfigyelésen alapul, hogy a disztributív algoritmusok futásideje tipikusan a gráf átmérőjétől függ, hiszen az eljárások szekvenciálisan fedezik fel az utakat, lépésenként egyetlen új pontot megismerve. Problémák tipikusan a súlyozottan rövid, de súlyozás nélküli értelemben (vagyis lépésszámban) hosszú utakkal vannak. Az alapötlet az, hogy a lépésszámban hosszú utakat véletlenszerű mintavétellel törjük meg. Véletlenszerűen ki fogunk választani $\Theta(\sqrt{n} \cdot \log n)$ pontot, amik a gráf *gerincét* vagy *vázát*(skeleton) adják majd. Egy adott csúcs a következőképpen irányít: ha a célállomás $\mathcal{O}(\sqrt{n})$ lépésre van tőle, akkor közvetlenül, ha a célállomás ennél messzebb, akkor pedig közvetetten, először ugyanis egy közeli vázponthoz irányít, ahonnan a csomag megtalálja az útját egy, a célállomáshoz közeli vázponthoz, ahonnan pedig könnyen tud navigálni magához a célhoz.
Alapvetően a következő dolgokra van szükségünk: a rövid távon navigáló short-range scheme-re, a vázpontok kiválasztására, a hosszú távon navigáló long-distance routing módszerre, illetve a rövid- és hosszútávú irányítás összekapcsolására. Mindezek előtt azonban vezessünk be néhány alapvető definíciót.

4.1. Definíciók

A következő definíciók során $G(V, E)$ egy irányítatlan gráf, csúcshalmaza V, élhalmaza E. Értelmezünk továbbá a gráfon egy $W : E \to \mathbb{N}$ költségfüggvényt. Legyenek $v, u \in V$ csúcsok, n pedig

a gráf csúcsainak száma.

4.1.1. Definíció. *Definiáljuk a következő élsúlyozatlan fogalmakat:*

- Nevezzük egy p út **lépéshosszának** az általa tartalmazott élek számát. Jelölés: $l(p)$.
- Nevezzük u és v csúcsok **lépéstávolságának** a következőt: $\mathrm{hd}(u,v) = \min\{l(p) \mid p \text{ út } u \text{ és } v \text{ között}\}$.
- A gráf **lépésátmérője** $\mathrm{HD} := \max_{u,\, v \in V}\{\mathrm{hd}(v,u)\}$.

4.1.2. Definíció. *Definiáljuk a következő, súlyozással kapcsolatos fogalmakat:*

- Egy p **út súlya** (vagy költsége) az út által tartalmazott élek súlyának összege. Jelölés: $\mathrm{W}(p)$.
- Az u és v pontok **W-súlyozott távolsága**, vagy **wd-súlyozott távolsága** $\mathrm{wd}(u,v) := \min\{\mathrm{W}(p) \mid p \text{ út } u \text{ és } v \text{ között}\}$.
- A G gráf **W-súlyozott átmérője** $\mathrm{WD} := \max\{\mathrm{wd}(u,v) \mid u,v \in V\}$.

4.1.3. Definíció. *Definiáljuk a következő fogalmakat, melyek vegyítik az élsúlyozatlan és az élsúlyozott fogalmainkat:*

- Adott $h \in \mathbb{N}$ és $u,v \in V$ csúcsok úgy, hogy $\mathrm{hd}(u,v) \leq h$. Ekkor u és v **h-súlyozott távolsága** az u-t és v-t összekötő, legfeljebb h élből álló utak közül a minimális távolságú, vagyis $\mathrm{wd_h}(u,v) := \min\{\mathrm{W}(p) \mid p \text{ út } u \text{ és } v \text{ között, és } l(p) \leq h\}$.
  Amennyiben $\mathrm{hd}(u,v) \geq h$, úgy $\mathrm{wd_h}(u,v) := \infty$.
- A G gráf **legrövidebb út átmérője** a legrövidebb utak élszámainak a maximuma, jelölés: SD.

4.1.4. Definíció. *Azt mondjuk, hogy egy esemény nagy valószínűséggel megtörténik, ha a komplementerének a valószínűségét beállíthatjuk $\frac{1}{n^c}$-nél kisebbre, bármilyen kívánt $c > 0$ konstansra.*

4.1.5. Definíció. *Azt mondjuk, hogy egy információ **lokálisan ismert** egy gráf egy csúcsánál, ha az adott csúcs címkéjéből kikövetkeztethető.*

Mielőtt rátérnénk a rövid- és hosszútávú irányításra, bemutatunk egy algoritmust, ami a későbbiekben alapvető építőkövünk lesz.

4.2. A BSP algoritmus

A BSP (*Bounded Shortest Paths*) egy módosított Bellman-Ford algoritmus. A szokásos $G(V, E)$ gráfon és a W költségfüggvényen kívül itt még három input adatunk van: h, ami egy távolság-paraméter; Δ, ami egy átfedés-paraméter; és egy $S \subset V$ csúcshalmaz, a forráspontok. Az algoritmus $\forall v \in V$-t felcímkéz oly módon, hogy a wd_h szerint legközelebbi Δ darab $s \in S$ forráspontba tudunk irányítani v-ből ezen címke alapján.

4.2.1. Megjegyzés. Magát a gráfot, illetve a költségfüggvényt továbbra sem soroljuk az inputok közé, mivel a *gráfon* futtatjuk az algoritmust. Azonban továbbra is feltesszük, hogy minden csúcs ismeri a rá illeszkedő éleket, és azok költségeit.

Input:

$h \in \mathbb{N}$, $\Delta \in \mathbb{N}$, $S \subseteq V$

Output:

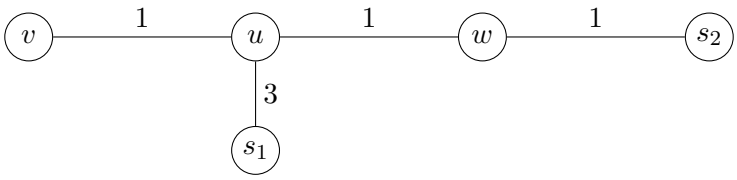
Minden $v \in V$-hez egy címkelista, mely $(\mathrm{wd_i}, s, \mathrm{next_v})$ címkékből áll, minden $i \in \{1, \ldots, h\}$-ra és Δ darab, az aktuális i által meghatározott $\mathrm{wd_i}$-legközelebbi $s \in S$ forráspontra, amelyre $\mathrm{wd_i}(v, s) < \infty$.

4.2.2. Definíció. *$L_v(t)$ a lista, amit $v \in V$ továbbküld a szomszédainak a t. iteráció után, vagyis a Δ darab wd_t-legközelebbi $s \in S$ forrás listája.*

Az algoritmus rövid leírása. A következő iterációt futtatjuk h-szor: Minden $v \in V$ (1) megkapja minden u szomszédjától $L_u(i)$-t, (2) a szomszédoktól kapott információk (és az odavezető élek költségeinek) függvényében frissíti a címkelistáját úgy, hogy csak a Δ legközelebbi forrásra vonatkozó címkéket vesz fel, (3) az új listát továbbküldi a szomszédainak.

Irányítás. A címkézés befejeztével, h lépés után szeretnénk elirányítani egy csomagot $v \in V$-ből a listájában szereplő valamely s forráspontba. Előfordulhat, hogy v címkéje szerint u felé kell haladjunk, $L_u(h)$ azonban nem tartalmazza s-t! Ezért nem elég $L_u(h)$-t tekinteni, hanem minden $0 \leq i \leq h : L_u(i)$-t kell, minden $u \in V$-re. A kibővített címkelistában már garantáltan benne lesz s. Általában, ha v és u közt i lépést hajtottunk végre, akkor $L_u(h - i)$-t használjuk.

4.2.3. Példa. Az alábbi gráfon $h = 2$, $\Delta = 1$, $S = \{s_1, s_2\}$ esetben h kör után $L_v(h)$-ban szerepelni fog s_1, $L_u(h)$-ban azonban nem, hiszen abban csak s_2 fog szerepelni. Ugyanakkor $L_u(1)$-ben szerepelni fog s_1.



4.2.4. Tétel. ([5], C. Lenzen, B. Patt-Shamir) *A BSP algoritmus minden $v \in V$-t felcímkéz oly módon, hogy v tud irányítani a Δ wd_h-legközelebbi forráspontba. Minden u csúcs, ami ezen a wd_h-legrövidebb úton van meg tudja állapítani a következő lépést a címkéjéből és a csomag által eddig megtett lépések számából. Az algoritmus legfeljebb $\mathcal{O}(\log n)$ méretű üzeneteket használ. Az eljárás időigénye $\mathcal{O}(\Delta h)$ lépés.*

Bizonyítás. □

4.3. Rövidtávú irányítás - A *short-range scheme*

A célunk a *short-range scheme*-ben minden csúcsot megcímkézni úgy, hogy el tudjon irányítani a hozzá $\widetilde{\Theta}(\sqrt{n})$ legközelebbi csúcsba. Vegyük észre, hogy a BSP algoritmus meghívása a gráfunkra $S := V$, $\Delta := \sqrt{n}$ és $h := \sqrt{n}$ paraméterekkel megoldja a problémát, csakhogy a 4.2.4 tétel miatt $\mathcal{O}(n)$ időben, aminél mi gyorsabb módszert szeretnénk. A probléma megoldására bevezetünk egy hierarchiát a csúcsok között.

4.3.1. A Hierarchia

Jelöljük L-lel a hierarchia szintjeinek számát, L értékét majd később határozzuk meg. A hierarchia szintjeit jelöljük $S_1, \ldots, S_L$-lel. $S_0 := V$, S_i pontjait pedig véletlenszerűen, független módon, egyenlő valószínűséggel választjuk ki S_{i-1}-ből $\forall\ 1 \leq i \leq L$ -re. Így egy egyre szűkülő halmazrendszert kapunk, ahol $S_i \subseteq S_{i-1}$ minden $0 \leq i \leq L$-re.

4.3.1. Definíció. *Minden $v \in V$-re és minden $0 \leq i \leq L$-re $Y_v(i)$ legyen a v-hez S_i-ben legközelebbi pont.*

4.3.2. Definíció. *Minden $v \in V$ csúcsra $H_v(i) := \{u \in S_{i-1} \mid wd(v,u) \leq wd(v, Y_v(i))\}$, vagyis az olyan S_{i-1} beli pontok, amik közelebb vannak v-hez, mint $Y_v(i)$.*

4.3.3. Definíció. *Egy $u \in S_i$ csúcsra, minden $1 \leq i \leq L$-re $C_u(i) := \{v \in V \mid Y_v(i) = u\}$, vagyis azon csúcsok halmaza, akikhez u a legközelebbi S_i-beli csúcs. Legyen $C_u(0) := \{u\}$.*

Input:

$n \in \mathbb{N}$, a csúcsok száma

$L \in \mathbb{N}$, a hierarchia szintjeinek száma

Output:

Minden $v \in V$-hez a csúcs szintjét jelző l_v úgy, hogy $v \in S_i \Leftrightarrow l_v \geq i$

A gráf egy olyan címkézése, mely alapján minden $i \in \{1, \ldots, L\}$-re minden $v \in V$-ből (nagy valószínűséggel) el lehet irányítani $Y_v(i)$-be és minden $H_v(i)$-beli csúcsba (nagy valószínűséggel) legrövidebb úton, továbbá minden $u \in S_i$-ből (nagy valószínűséggel) el lehet irányítani minden $C_u(i)$-beli csúcsba (nagy valószínűséggel) legrövidebb úton.

Az **output** jobban kifejtve azt jelenti, hogy minden v csúcs lokálisan ismeri $Y_v(i)$-t és $H_v(i)$-t, az oda vezető távolságokat és next pointereket, valamint az oda vezető úton minden csúcs képes folytatni az irányítást oly módon, hogy legrövidebb utat kapjunk (nagy valószínűséggel). Hasonló igaz minden $u \in S_i$ csúcsra is: $C_u(i)$-t lokálisan ismerik, csak úgy, mint az oda vezető távolságokat és next pointereket, és az oda vezető úton szintén minden csúcs tudja folytatni az irányítást oly módon, hogy legrövidebb utat kapjunk (nagy valószínűséggel).

4.3.2. Az algoritmus

A következő iterációt futtatjuk L-szer: az i. lépésben minden $v \in V$ csúcs talál egy utat $Y_v(i)$-be és $H_v(i)$ minden csúcsába. Ahhoz, hogy ezt végrehajtsuk, meghívjuk a BSP algoritmust S_{i-1} forráshalmazzal. Nézzük, milyen h_i és Δ_i paramétereket kell adjunk az algoritmusnak ahhoz, hogy nagy valószínűséggel a kívánt outputot kapjuk.
Jelölje p_i a valószínűségét annak, hogy egy pont bekerül S_i-be.

4.3.4. Állítás. *Nagy valószínűséggel minden $v \in V$-nek van egy S_i-be eső szomszédja a hozzá hd-legközelebbi $\mathcal{O}(\frac{\log n}{p_i})$ csúcs között.*

Bizonyítás. □

4.3.5. Következmény. Válasszuk h_i-t úgy, hogy $h_i \in \Theta(\frac{\log n}{p_i})$.

4.3.6. Állítás. *Azon S_{i-1}-beli csúcsok várható száma, melyek egy rögzített v csúcs h_i darab hd-legközelebbi szomszédjai közt vannak $\mathcal{O}(\log n \frac{p_{i-1}}{p_i})$.*

Bizonyítás. Annak a valószínűsége, hogy u S_{i-1}-beli, p_i. Ezért ezen csúcsok várható száma $p_{i-1} \cdot h_i = \mathcal{O}(\log n \cdot \frac{p_{i-1}}{p_i})$. □

4.3.7. Következmény. Válasszuk Δ_i-t úgy, hogy $\Delta_i \in \Theta(\log n \cdot \frac{p_{i-1}}{p_i})$.

A 4.2.4 tétel miatt a BSP algoritmus futásideje az i. iterációban $T_i \in \mathcal{O}(h_i \cdot \Delta_i) \subseteq \widetilde{\mathcal{O}}(\frac{p_{i-1}}{p_i^2})$. Mivel az a célunk, hogy ez a futásidő $\widetilde{\mathcal{O}}(\sqrt{n})$-es legyen, ezért $T_i := \sqrt{n}$ és $p_0 := 1$.

4.3.8. Állítás. *A* $\frac{p_{i-1}}{p_i} = T_i$ $(i \in \{1, \ldots, L\})$ *rekurzív egyenletrendszer megoldása* $p_0 = 1$ *és* $T_i = \sqrt{n}$ *értékekre* $p_i = n^{-\frac{2^i-1}{2^{i+1}}}$.

Bizonyítás. Az állítást i-re való indukcióval látjuk be. Először tekintsük az $i = 1$ esetet. Ekkor a $\sqrt{n} = \frac{1}{p_1^2}$ egyenletet kell megoldanunk, melyet átrendezve kapjuk, hogy $p_1 = n^{-\frac{1}{4}}$. Tegyük fel, hogy az állítást tudjuk $i = k$-ra. Ekkor

$$\sqrt{n} = \frac{n^{-\frac{2^k-1}{2^{k+1}}}}{p_{k+1}^2}$$

$$p_{k+1}^2 = n^{-\frac{2^k-1}{2^{k+1}}} \cdot n^{-\frac{1}{2}}$$

$$p_{k+1}^2 = n^{-\frac{2^{k+1}-1}{2^{k+1}}}$$

$$p_{k+1} = n^{-\frac{2^{k+1}-1}{2^{k+2}}}$$

Ezzel az állítást beláttuk. □

4.3.9. Következmény. A BSP algoritmusok paramétereit válasszuk a következőképpen: $h_i \in \Theta(n^{\frac{1}{2}-\frac{1}{2^{i+1}}} \log n)$, illetve $\Delta_i \in \Theta(n^{\frac{1}{2^{i+1}}} \log n)$ minden $i \in \{1, \ldots, L\}$-re.

4.3.10. Következmény. Az S_i forráshalmazok várható méretei nagy valószínűséggel a következők: $|S_i| \in \Theta(n^{\frac{1}{2}+\frac{1}{2^i}})$ minden $i \in \{1, \ldots, L\}$-re.

4.3.11. Állítás. ([5]) *Ahhoz, hogy* $S_L \in \Theta(\sqrt{n})$ *nagy valószínűséggel, elegendő* $L = \log\log n$ *szint.*

Bizonyítás. □

Az előző állítás miatt innentől feltételezzük, hogy $L \in \mathcal{O}(\log\log n)$.

4.3.12. Megjegyzés. Vegyük észre, hogy az algoritmusunk nem fogja lehetővé tenni a fordított irányba való irányítást, vagyis hogy $Y_i(v)$-ből elirányítsunk $C_v(i)$-be. Ennek a megvalósításához a [6]-ban ismertetett algoritmust használjuk, amit itt nem részletezünk. Ezen algoritmus $\widetilde{\mathcal{O}}(h_i)$ körben megkonstruálja a címkéket és irányító táblákat, amik megadnak egy $Y_v(i)$ gyökerű fát, melynek segítségével el tudunk irányítani $Y_v(i)$-ből $C_v(i)$-be. Egy címke mérete $\mathcal{O}(\log n)$, a táblák mérete pedig $\widetilde{\mathcal{O}}(1)$.

4.3.3. Az algoritmus - tulajdonságok

4.3.13. Lemma. ([5]) *Az $i \in \{1, \ldots, L\}$ szinteken a következő tulajdonságok nagy valószínűséggel igazak maradnak:*
(1) S_i egy véletlen részhalmaza S_{i-1}-nek, ahova minden S_{i-1}-beli csúcsot függetlenül, egyenlő valószínűséggel választunk be, és $P(v \in S_i) = p_i$ és $P(v \in S_i \mid v \in S_{i-1}) = \frac{p_i}{p_{i-1}}$.
(2) Minden $v \in V$ csúcsra el tudunk irányítani v-ből $Y_v(i)$-be wd-legrövidebb úton.
(3) Minden $v \in V$ csúcsra $Y_v(i)$ és $wd(v, Y_v(i))$ megadható v címkéjéből.
(4) Minden $u \in S_i$ csúcsra bármely $w \in C_u(i)$ csúcs címkéjének ismeretében tudunk u-ból w-be irányítani wd-legrövidebb úton.
(5) Minden $v \in V$ csúcsra $H_v(i)$ lokálisan ismert v-nél, és bármely $u \in H_v(i)$ csúcsba tudunk irányítani v-ből, legrövidebb úton, aminek a költsége ismert v-nél.

Bizonyítás. □

4.3.14. Lemma. *Ha ismerjük a hierarchiát, ami a fenti tulajdonságokkal rendelkezik, akkor w címkéje alapján tudunk irányítani v-ből w-be, amennyiben*

$$w \in \bigcup_{1 \le i \le L,\, u \in H_v(i)} C_u(i-1)$$

Bizonyítás. Vagyis v-ből el tudunk irányítani az olyan w csúcsokba, akikhez a legközelebbi pont valamelyik $i-1$. szinten egy olyan u pont, ami benne van $H_v(i)$-ben, vagyis valamelyik szinten olyan S_{i-1}-beli, ami közelebb van v-hez, mint $Y_v(i)$. Az irányítás miként je a fenti tulajdonságokból könnyen látható: (5) miatt v-ben lokálisan ismert az összes olyan u, aki szóba jöhet köztes pontnak, ebben az u-ban pedig lokálisan ismert lesz $Y_u(i)$ a (3)-as tulajdonság miatt. A v-ből w-be vezető út tehát a v-ből u-ba, illetve az u-ból w-be vezető utak egymásutánja lesz. □

4.3.4. Stretch és futásidő

4.3.15. Lemma. ([5]) *Tegyük fel, hogy $v, w \in V$ csúcsokra és $1 \le j \le L$-re*

$$w \notin \bigcup_{i=1}^{j} \bigcup_{u \in H_v(i)} C_u(i-1).$$

Ekkor a) $wd(v, Y_v(j)) \le (2j-1) \cdot wd(v, w)$, valamint b) $wd(w, Y_w(j)) \le 2j \cdot wd(v, w)$.

Bizonyítás. □

4.3.16. Állítás. *Legyenek $v, w \in V$ csúcsok, $1 \le i_0 \le L$ pedig a legkisebb olyan, amire $Y_w(i_0 - 1) \in H_v(i_0)$. Ekkor $wd(v, Y_w(i_0-1)) + wd(Y_w(i_0-1), w) \le (4i_0 - 3) \cdot wd(v, w) \in \mathcal{O}(L \cdot wd(v, w))$.*

Bizonyítás.

$$\mathrm{wd}(v, Y_w(i_0-1)) + \mathrm{wd}(Y_w(i_0-1), w) \leq \mathrm{wd}(v,w) + 2 \cdot \mathrm{wd}(Y_w(i_0-1), w) \leq$$

$$\leq \mathrm{wd}(v,w) + 4 \cdot (i_0-1) \cdot \mathrm{wd}(v,w) = (4i_0-3) \cdot \mathrm{wd}(v,w).$$

Az első átalakítás a háromszög-egyenlőtlenségből következik, a második [4.3.15]-ből. Az állítást ezzel beláttuk. □

4.3.17. Tétel. ([5], C. Lenzen, B. Patt-Shamir) *Adott $1 \leq L \leq \log\log n$-re az L-szintű rövidtávú irányításhoz használt irányító táblák és címkék megkonstruálhatóak*

$$\mathcal{O}(L \cdot (\sqrt{n})^{\frac{2^L}{2^L-1}} \cdot \log^2 n) \subset \widetilde{\mathcal{O}}((\sqrt{n})^{\frac{2^L}{2^L-1}})$$

körben úgy, hogy a csúcsok címkéi $\mathcal{O}(L \log n)$ bitesek.

4.4. Hosszútávú irányítás - *Long-Distance Routing*

Az algoritmus bevezőjében már ismertettük a koncepciót, miszerint a távoli pontok közti irányítás a gráf *csontvázán* fog zajlani. Ennek a pontos miként jét fogjuk leírni ebben a részben, azt azonban érdemes előrevetíteni, hogy a csontvázat a rövidtávú irányítás során megismert legfelső hierarchia-osztály, S_L fogja adni.

4.4.1. Definíció. (Csontváz gráf - Skeleton graph) *Adott, W súlyfüggvénnyel, V csúcshalmazzal és E élhalmazzal rendelkező $G = (V, E, W)$ gráfra, $S \subseteq V$ csúcshalmazra és $h \in \mathbb{N}$-re a h lépésű, S csontvázú súlyozott $G_{S,h} = (S, E_{S,h}, W_{S,h})$ gráfot a következőképpen definiáljuk:*
$E_{S,h} := \{\{v,w\} \mid v,w \in S$ és $hd_G(v,w) \leq h\}$, valamint
$\forall\ \{v,w\} \in E_{S,h}$-ra $W_{S,h}(v,w) := wd_h(v,w)$, vagyis $W_{S,h}$ a h-súlyozott távolság v és w között G-ben.

A definícóban $hd_G(u,v)$-vel jelölt távolság a két pont lépéstávolsága G-ben, vagyis a két pontot összekötő utak közül a minimális élszámúnak az élszáma. Emlékeztetünk továbbá, hogy a 4.1.3 definíció alapján a h-súlyozott távolság a két pontot összekötő, legfeljebb h élből álló utak közül a minimális költségű. Ezek alapján észrevehető, hogy $E_{S,h}$ nem részhalmaza E-nek, és a csontvázgráfban megjelenő élsúlyok sem feltétlenül fordulnak elő G-ben élsúlyként.
Azonban, amennyiben S véletlen részhalmaza V-nek, valamint $h \in \Omega(\frac{n \cdot \log n}{|S|})$, akkor a $G_{S,h}$-beli távolságok nagy valószínűséggel megegyeznek a G-beli távolságokkal. Ez azt jelenti, hogy G-ben elegendő volna $\mathcal{O}(\frac{n \cdot \log n}{|S|})$ élből álló utakat tekinteni, ha legrövidebb utat keresünk. Ezt a következő lemmában foglaljuk össze:

4.4.2. Lemma. ([5]) *Legyen S_R egy csúcsokból álló véletlen halmaz, melynek tagjait egyenlő valószínűséggel választjuk ki: $P(v \in S_R) = p$ valamilyen adott p-re. Valamint legyen $S \supseteq S_R$. Ha $p \geq c \cdot \frac{\log n}{h}$ egy elegendően nagy c konstansra, akkor nagy valószínűséggel $wd_{S,\,h}(u,v) = wd(u,v)$ $\forall\ u, v \in S$-re.*

Bizonyítás. □

Célunk megkonstruálni $G_{S,\,h}$-t, és kiszámolni a legrövidebb utakat minden pontjából minden pontjába. Két problémába ütközünk: az egyik, hogy $G_{S,\,h}$ élei virtuálisak – egy legfeljebb h élből álló legrövidebb útnak felelnek meg G-ben, a másik pedig, hogy $G_{S,\,h}$-nak rengeteg, akár $\Omega(|S|^2)$ éle lehet.
A két problémát egyszerre fogjuk megoldani. Miközben a csontváz-gráf éleit és távolságait számoljuk, ritkítani fogjuk a csontvázgráf éleit. Ha ez kész van, akkor a csontváz kellően ritka lesz ahhoz, hogy teljes topológiáját megismerhesse *minden csúcs V-ben.*

4.4.3. Definíció. *Legyen $H = (V, E)$ egy súlyozott gráf W súlyfüggvénnyel, legyen továbbá $k \geq 1$. H egy súlyozott k-spannerje egy súlyozott $H' = (V, E', W')$ gráf, ahol $E' \subseteq E$, $W'(e) = W(e)$ $\forall\ e \in E'$ élre, valamint $wd_{H'}(u,v) \leq k \cdot wd_H(u,v)$ $\forall\ u, v \in V$, ahol $wd_{H'}$ és wd_H a H'-beli illetve a H-beli súlyozott távolságokat jelentik.*

4.4.1. A Baswana-Sen spannerkonstrukciós algoritmus

Ebben a részben ismertetjük a Baswana-Sen algoritmust, amire a hosszútávú irányításunk épülni fog. Az eljárás részletes leírása megtalálható [7]-ben. Alapötlete az, hogy a csúcsokat az algoritmus során csoportokba (ún. *clusterekbe*) rendezzük. A bemenet egy $H = (V_H, E_H, W_H)$ gráf és egy $k \in \mathbb{N}$ paraméter.
Az algoritmus úgynevezett *fázisokra* oszlik. Az első fázisban minden csúcs egy egyelemű cluster, ezeket jelölje R_1. A későbbiekben $i \in \{1, \ldots, k-1\}$-re az i. fázisban R_i minden tagját véletlenszerűen, függetlenül, azonos valószínűséggel megjelöljük. A megjelölt clusterekből fog állni R_{i+1}.

Algorithm 2: A Baswana-Sen algoritmus

1. $R_1 := \{\{v\} \mid v \in V_H\}$, vagyis minden csúcs egy egyelemű cluster;
2. **for** $k = 1, \ldots, k-1$ **do**
  a. R_i minden clusterét jelöljük meg független módon, véletlenszerűen $|V_H|^{\frac{-1}{k}}$ valószínűséggel. $R_{i+1} :=$ {megjelölt clusterek} ;
  b. if *v egy jelöletlen cluster csúcsa* **then**
    Q_v legyen azon élek halmaza, amik legrövidebb élek v és a v-vel szomszédos clusterek között ;
    if *Ha v egyik szomszédos clustere sem jelölt* **then**
      Q_v összes élét adjuk hozzá a spannerhez;
    else
      Legyen u a v-hez legközelebbi olyan pont, ami jelölt clusterben van. Ekkor v csatlakozik az u-t tartalmazó clusterhez, majd hozzáadja a spannerhez $\{v, u\}$-t, illetve Q_v olyan éleit, amik $\{v, u\}$-nál rövidebbek;
3. Minden csúcs hozzáad a spannerhez minden olyan élt, ami legrövidebb él közte és egy szomszédos $X \in R_k$ cluster között ;

4.4.4. Tétel. ([7], Baswana, Sen) *Adott $H(V_H, E_H)$ súlyozott gráfra és $k \geq 1$ paraméterre a fenti algoritmus kiszámolja H egy $(2k-1)$-spannerjét, aminek nagy valószínűséggel $\mathcal{O}(k \cdot |V_H|^{1+\frac{1}{k}} \cdot \log n)$ éle van.*

Bizonyítás. □

4.4.2. Az LDC algoritmus

Ebben a részben ismertetjük az LDC (vagyis Long Distance Construction) algoritmust. Alapja az imént bemutatott Baswana-Sen algoritmus. Azonban a mi esetünkben az élek, amiket a fenti algoritmus **2/b** és **3** lépéseiben használunk, egy-egy legrövidebb útnak felelnek meg az eredeti gráfban (lásd az 4.4.1 definíciót). A fenti algoritmus tehát módosításra szorul, hiszen így nem elegendő szomszédos clustereket keresni. A módosítás nem triviális, hiszen a Baswana-Sen algoritmusban minden csúcstól legfeljebb egy lépéstávolságra lévő clustereket tekintettünk (vagyis szomszédosakat), viszont amikor áttérünk élekről legrövidebb utakra, az éleken értelmezett "szomszédos" fogalom már nem feleltethető meg olyan könnyen egy, a legrövidebb utakon értelmezett hasonló fogalomnak. Erre a problémára a következő lemma ad megoldást.

4.4.5. Lemma. ([5]) *A Baswana-Sen algoritmus outputja nagy valószínűséggel nem változik, amennyiben a 2/b és 3 lépéseiben minden csúcs csak a $c \cdot |V_H|^{\frac{1}{k}} \cdot \log n$ legközelebbi clusterhez*

vezető legrövidebb éleket tekinti (kellően nagy $c > 0$ konstansra).

Bizonyítás. □

4.4.6. Következmény. A BSP' algoritmust $\Delta \in \mathcal{O}(|S|^{\frac{1}{k}} \cdot \log n)$ átfedés paraméterrel hívjuk meg. Az 4.4.2 lemma következményeként a h távolságparaméter legyen $h \in \mathcal{O}(n \cdot \frac{\log n}{|S|})$.

A **2/b** és a **3** lépésekben tehát legrövidebb utakat keresünk a legközelebbi $c \cdot |V_H|^{\frac{1}{k}} \cdot \log n$ clusterbe. Ennek implementálásához egy módosított BSP algoritmust fogunk használni.
Mivel legközelebbi clustereket szeretnénk találni, nem pedig legközelebbi csúcsokat, ezért a hagyományos BSP-algoritmus nem lesz megfelelő: hiszen az egy adott csúcshoz a legközelebbi Δ darab forráspontot találja meg, tehát ha módosítatlanul futtatnánk, a clusterbeli pontokkal, mint forráspontokkal, akkor előfordulhatna, hogy csak egyetlen clustert talál meg (abban pedig Δ forráspontot). Ennek kiküszöbölésére a forráscsúcsok azonosítására használt *forrás-azonosítót* ugyanarra állítjuk egy adott clusteren belül.

4.4.7. Definíció. *Egy $v \in V$ csúcsra source(v) a csúcs forrásazonosítója. Értéke $\perp$, ha a csúcs nem forrás; v, ha forrás a saját azonosítójával; w, amennyiben forrás egy $v \neq w \in V$ csúcs azonosítójával.*

Célszerű úgy elképzelni, mintha egy X cluster összes csúcsát átneveznénk x-re. Ezután az algoritmus egy clustert csak egyetlen egyszer fog feljegyezni magának, és így a Δ legközelebbi *clustert* fogja megtalálni, nem pedig a Δ legközelebbi csúcsot (lásd az 4.2 részt a BSP algoritmus leírásáért).
Ugyanakkor az irányítás végrehajtása végett szükségünk van az út végpontjára, ezért a clusterbeli csúcsoknak megtartjuk az eredeti azonosítóját is, és ezeket is feljegyezzük a címkelistákba. Tehát a BSP-algoritmus $L_v(i)$ információhármasai (lásd az 4.2.2 definíciót) helyett $L'_v(i)$ információnégyeseket továbbítunk. Ez az algoritmust végrehajtását és futásidejét érdemben nem befolyásolja. A módosított algoritmust nevezzük BSP'-nek.

4.4.8. Megjegyzés. Egy $(d, f, u, w) \in L'_v$ alakú címke jelentése: az f clusterben található w csúcsba vezető d súlyú út következő csúcsa u. Az algoritmusben továbbá szükségünk van arra, hogy megkülönböztessük egymástól a jelölt és jelöletlen clustereket, ezért a címkéink a következőképpen fognak kinézni: $(d, (f, b), u, w)$, ahol $b = 1$, ha f jelölt cluster, $b = 0$, ha jelöletlen.

Először az EDGES algoritmust mutatjuk be, melyet szubrutinként fog használni az LDC-algoritmus. Az EDGES-t használjuk a Baswana-Sen spannerkonstrukció **2/b** és **3** lépéseinek megvalósításához.

4.4.9. Definíció. *Egy $X \subseteq V$ clusterhez tartozó csúcsot a **cluster vezetőjének** vagy **clustervezérnek** nevezünk, amennyiben minden $x \in X$ csúcs az ő forrásazonosítóját használja.*

4.4.10. Definíció. *Definiáljunk egy F függvényt, ami megadja egy $v \in V$ csúcs clusterének vezetőjét. $F : V \to V \cup \{\perp\}$, $F(v) = w$, amennyiben v clusterének vezetője w, valamint $F(v) = \perp$, amennyiben v nem tartozik clusterhez.*

Algorithm 3: EDGES:

input :
$F : V \to V \cup \{\perp\}$; $R \subseteq V$; $h \in \mathbb{N}$; $\Delta \in \mathbb{N}$
output:
$E_+ \subseteq E$, a spannerhez hozzáadandó élek
$W_+ : E_+ \to R^+$, ezen élek költségei

```
1  foreach w ∈ V do
2      if F(w) ∉ R ∪ {⊥} then
3          source(w) := (F(v), 0) ;
4      else
5          if F(w) ∈ R then
6              source(w) := (F(v), 1) ;
7          else
8              source(w) := ⊥ ;
9  L_v := BSP'(h, Δ, source) ;
10 E_+ := ∅ ;
11 if F_v ∉ R ∪ {⊥} then
12     L_v := L_v \ {(0, (F(v), 0), v, v)} ;          // körök eltávolítása
13     foreach (d, (f, b), u, w) ∈ L_v do
14         broadcast (d, {v, w}) ;
15         E_+ := E_+ ∪ {v, w} ;
16         W_+({v, w}) := d ;
17         if f ∈ R then
18             break ;
19 return (E_+, W_+)
```

4.4.11. Megjegyzés. Az algoritmus inputként kapott négy paraméter közül R (a jelölt clusterek), h és Δ globálisan ismert, vagyis minden csúcs használhatja őket a lokális műveleteknél. A negyedik paraméter, az imént ismertetett F lokálisan ismert, vagyis minden v csúcs ismeri $F(v)$ értékét.

4.4.12. Megjegyzés. Az algoritmus során $F(v)$ helyett valójában $(F(v), 0)$, illetve $(F(v), 1)$ párosokat használunk, ezek célja, hogy meg tudjuk különböztetni a jelölt clustereket a jelöletlenektől.

4.4.13. Megjegyzés. Az algoritmus során feltesszük, hogy a rendelkezésünkre áll egy 'broadcast' utasítás, ami bizonyos információt elérhetővé tesz minden csúcs számára. Ennek a parancsnak a végrehajtásakor az adott információt minden csúcs minden élén továbbküldi. Könnyen látható, hogy ahhoz, hogy egy ilyen információ minden csúcshoz eljusson, HD körre van szükség. Részletes leírás [5]-ben található.

4.4.14. Megjegyzés. Fontos megjegyezni, hogy a BSP illetve BSP' algoritmusok távolság szerinti növekvő sorba rendezik az outputjaikat (vagyis a csúcsok címkelistáit). Ennek megfelelően az edges algoritmus 13. sorában úgy értjük a feltételt, hogy növekvő sorrendben haladunk a címkéken. Így az algoritmus pontosan a Baswana-Sen algoritmus **2/b** illetve **3**-as lépéseit hajtja végre, hiszen a címkelistában első jelölt cluster a legközelebbi jelölt cluster lesz, az ez előtti clusterek pedig a nála közelebbi jelöletlenek.

Most pedig lássuk a hosszútávú konstrukciós (LDC) algoritmust. Az eljárás 13. sorában használt $E(i)$ és $W(i)$ pusztán jelölések, az EDGES algoritmus outputja annak i. meghívására.

4.4.15. Definíció. *Egy $v \in V$ csúcsra **marked(v)** eldönti, hogy v jelölt csúcs-e, azaz jelölt clusterhez tartozik-e.*

Algorithm 4: Az LDC algoritmus

input : $G(V,E)$ gráf W költségfüggvénnyel; $S \subseteq V$, a csontvázpontok halmaza; $k \in \mathbb{N}$, paraméter

output: $E_{h,k}$, a csontváz-gráf spanner-élei; $W_{h,k}$, az ezekhez tartozó költségek

$R_1 := \{\{w\} \mid w \in S\}$;
broadcast R_1 ;
foreach $w \in V$ **do**
    if $w \in R_1$ **then**
        $F_1(w) := w$
    else
        $F_1(w) := \perp$
$h := c \cdot n \cdot \frac{\log n}{|S|}$;
$\Delta := c \cdot |S|^{\frac{1}{k}} \log n$;
for $i := 1$ **to** $k-1$ **do**
    $R_{i+1} :=$ R_i-ből véletlenszerűen, egyenlő valószínűséggel választott csúcsok halmaza, melynek mérete legyen $|S|^{1-\frac{i}{k}} = \frac{|R_i|}{|S|^{\frac{1}{k}}}$;
    broadcast R_{i+1} ;
    $(E(i), W(i)) := \text{edges}(F_i, R_{i+1}, h, \Delta)$;
    foreach $w \in V$ **do**
        if $F_i(w) \in R_{i+1}$ **then**
            $F_{i+1}(w) := F_i(w)$;
        else
            $E_w := \{w\text{-vel szomszédos élek } E(i)\text{-ben}\}$;
            if $E_w \neq \emptyset$ **then**
                Legyen $\{(w,u)\} \in E_w$ a legnagyobb költségű él E_w-ben ;
                if $marked(u)$ **then**
                    $F_{i+1}(w) := F_i(u)$;
            else
                $F_{i+1}(w) := \perp$;
        broadcast F_{i+1} ;
    $(E(k), W(k)) := \text{edges}(F_k, \emptyset, h, \Delta)$;
    foreach $e \in \bigcup_{i=1}^{k} E(i)$ **do**
        $W_{h,k}(e) := W(k)(e)$;
    $E_{h,k} := \bigcup_{i=1}^{k} E(i)$;
    broadcast $E_{h,k}$, $W_{h,k}$;

4.4.16. Lemma. ([5]) *Tegyük fel, hogy a 4-as algoritmus inputjában szereplő S tartalmaz egy egyenlő valószínűséggel, független módon kiválasztott részhalmazt, S_R-t és legyen $h(S_R) := c \cdot n \cdot \frac{\log n}{|S_R|}$ egy elég nagy c konstansra. Ekkor a következők nagy valószínűséggel teljesülnek:*
(i) A 4-as algoritmus kiszámítja a $G_{S,h(S_R)}$ csontváz gráf egy $(2k-1)$ spannerjét, amit minden csúcs ismer, és aminek $\widetilde{\mathcal{O}}(|S|^{1+\frac{1}{k}} \cdot \log n)$ éle van.
(ii) Az S-beli csúcsok súlyozott távolsága megegyezik G-ben és $G_{S,h(S_R)}$-ben.
(iii) Az algoritmus leáll $\widetilde{\mathcal{O}}(\frac{n}{|S_R|^{1-\frac{1}{k}}} + |S|^{1+\frac{1}{k}} + HD)$ kör után.

Bizonyítás. □

4.4.17. Lemma. ([5]) *Legyen $\{s,t\}$ egy él a 4-as algoritmus által konstruált $G_{S,h(S_R)}$ spannerben. Ekkor nagy valószínűséggel a legfeljebb $h(S_R)$ élből álló legrövidebb $s-t$ úton minden csúcs meg tudja állapítani ennek az útnak a következő csúcsát és a hátralévő részút költségét $\widetilde{\mathcal{O}}(\frac{n}{|S_R|} + |S|^{1+\frac{1}{k}})$ kör alatt.*

Bizonyítás. □

4.4.18. Tétel. ([5], C. Lenzen, B. Patt-Shamir) *Tegyük fel, hogy az S csúcshalmaznak részhalmaza egy $S_R \subseteq V$ halmaz, mely véletlenszerűen, egyenlő valószínűséggel, függetlenül lett kiválasztva. Tegyük fel továbbá, hogy $k \in \{1, \ldots, \log n\}$. Ekkor nagy valószínűséggel az algoritmus megkonstruálja az S-beli csúcsok közti, legfelejebb $(2k-1)$ stretch-ű irányításhoz szükséges irányító táblázatot $\widetilde{\mathcal{O}}(\frac{n \cdot |S|^{\frac{1}{k}}}{|S_R|} + |S|^{1+\frac{1}{k}} + HD)$ kör alatt.*

Bizonyítás. □

4.5. Az alkotóelemek összeillesztése

Ebben a részben a rövid- és hosszútávú konstrukciós algoritmusokat összekapcsoljuk úgy, hogy a short-range scheme-ben megismert S_L-t megfeleltetjük az LDC algoritmus által használt csontváznak. Az eddigi eredményeket felhasználva pedig megadunk egy végső irányító algoritmust. Az algoritmus alatt v-ből szeretnénk irányítani w-be. A rövidtávú irányítás paramétere L, a hosszútávúé k. Az irányítás végrehajtásához v megkapja w címkéjét, $\lambda(w)$-t.

1 **if** $Y_v(i) = Y_w(i)$ *valamely* $i \in \{1, \ldots, L\}$*-re* **then**
2 Ekkor el tudunk irányítani $Y_v(i)$-ből w-be a 4.3.12 megjegyzés szerint. Ekkor d legyen v és w távolsága a megjegyzésben említett algoritmus (lásd [6]-et) által megkonstruált fán.
3 **else**
4 **if** $Y_w(i-1) \in H_v(i)$ **then**
5 $d_i := \mathrm{wd}(v, Y_w(i-1)) + \mathrm{wd}(Y_w(i-1), w)$
6 **else**
7 $d_i := \infty$
8 Legyen $S_v \subseteq S_L$ azon csontvázcsúcsok halmaza, melyekre vonatkozóan v címkéje tartalmaz információt. Legyen egy $s \in S_v$ csúcsra d_s a v címkelistájában tárolt távolság. ;
9 Legyen wd^k a csontváz spannerének költségfüggvénye. Legyen $d_{L+1} := \min_{s \in S_v}\{d_s + \mathrm{wd}^k(s, Y_w(L)) + \mathrm{wd}(Y_w(L), w)\}$. ;
10 Végül v kiszámítja a $d := \min_{i \in \{1,\ldots,L+1\}}\{d_i\}$ távolságot.
11 A kiszámított d távolságnak megfelelően irányít v, vagyis a d-nek megfelelő címkéjéből veszi a *next* pointert.

4.5.1. Megjegyzés. A 4.3.13 lemma biztosítja, hogy v a $\lambda(w)$ címke ismeretében elegendő információval rendelkezik, hogy a szükséges számításokat lokálisan elvégezze. Ezt úgy értjük, hogy az algoritmusok által konstruált címkelistáján és a $\lambda(w)$ címkelistán kívül más információt nem használ.

4.5.2. Megjegyzés. A 4.3.13 lemma (2)-es pontja miatt $Y_v(L) \in S_v$, vagyis S_v nem üres (nagy valószínűséggel).

4.5.3. Lemma. ([5]) *Legyen L és k tetszőleges rögzített paraméterek, előbbi az LDC-algoritmusé, utóbbi a short-range-scheme-é. Tetszőleges v csúcsra és $\lambda(w)$ címkelistára számítsuk ki a fenti*

algoritmussal d-t, és a next pointer által meghatározott csúcsot jelöljük u-val, az u-n $\lambda(w)$-vel futtatott algoritmus által megadott távolságot pedig d'-vel. Ekkor nagy valószínűséggel $d \leq (8kL-1) \cdot wd(v,w)$ és $d' \leq d - wd(v,u)$.

Bizonyítás. □

4.5.4. Következmény. Mivel az élsúlyok pozitívak, ezért az előző lemma második feléből rögtön következik, hogy az irányítás aciklikus és idővel véget ér.

4.5.5. Tétel. ([5], C. Lenzen, B. Patt-Shamir) *Legyen $\frac{1}{2} \leq \alpha \leq 1$ adott. Definiáljuk k-t a következőképpen: $k := \left\lceil \frac{1}{2\alpha - 1} \right\rceil$, ha $\alpha \geq \frac{1}{2} + \frac{1}{\log n}$, máskülönben $k := \log n$. Ekkor a fenti algoritmus nagy valószínűséggel megkonstruálja az irányító táblákat az irányításhoz és távolság-approximációhoz $\mathcal{O}(\log n)$ méretű üzenetekkel $\widetilde{\mathcal{O}}(n^\alpha + HD)$ kör alatt. A távolságok stretche $\rho_\alpha = 8k\lceil \log(k+1) \rceil - 1$, a címkék mérete $\mathcal{O}(\log(k+1)\log n)$.*

Bizonyítás. □

4.5.6. Állítás. ([5]) *A fenti algoritmus által egy $v \in V$ csúcsnál kiszámított címkelista mérete $\widetilde{\mathcal{O}}(n^\alpha)$.*

Bizonyítás. □

4.5.7. Állítás. ([5]) *Az algoritmus stretche $\rho_\alpha \in \mathcal{O}(\log n \log\log n)$, ha $\alpha = \frac{1}{2}$, és $\rho_\alpha \in \mathcal{O}(1)$, ha $\alpha > \frac{1}{2}$.*

Bizonyítás. □

Az előző állítás tehát azt mondja, hogy ha $\alpha > \frac{1}{2}$, vagyis megelégszünk $\widetilde{\mathcal{O}}(\sqrt{n} + HD)$-nél kissé rosszabb futásidővel, akkor az algoritmus stretche nem függ n-től.

Irodalomjegyzék

[1] Andrew S. Tanenbaum, *Számítógéphálózatok*, Panem, 2004, 391-401 oldalak.

[2] C. Hedrick, *Routing Information Protocol*, Rutgers University, 1988, https://tools.ietf.org/html/rfc1058

[3] Fabian Kuhn, Rotem Oshman, *Dynamic Networks: Models and Algorithms*, ACM SIGACT News, 2011.

[4] Fabian Kuhn, Nancy Lynch, Rotem Oshman, *Distributed Computation in Dynamic Networks*, STOC'10, 2010.

[5] Christoph Lenzen, Boaz Patt-Shamir, *Fast Routing Table Construction Using Small Messages*, 2012, http://arxiv.org/abs/1210.5774v2

[6] Mikkel Thorup, Uri Zwick, *Compact routing schemes*, 2001

[7] S. Baswana and S. Sen, *A Simple and Linear Time Randomized Algorithm for Computing Sparse Spanners in Weighted Graphs*, Random Structures and Algorithms, 2007, 532-563 oldalak.